Creating a simulation to assess performance of MAC-level opportunistic forwarding schemes in linear cluster topologies

Daniel Hope
76918328

Supervisor:

Andreas Willig

16/10/2015

University of Canterbury

Abstract

The goal of this project was to create a software simulation of a wireless network. The wireless nodes should use a CSMA MAC protocol and should be able to have their back off distributions changed so the performance differences can be studied for future research. The OMNeT++ and MiXiM frameworks were used to provide an environment to build the simulation in. However during implementation bugs in the MiXiM framework slowed down progress and has left the simulation with errors. The bit error rate is not being calculated correctly is one such error. Before the simulation could be used for research purposes these issues need to be resolved. The design of the wireless devices allows for the back off distribution to be changed easily between simulation runs. Having it specified in the configuration file allows for the distributions to be assigned based on a nodes group number, or even assigned individually.

Contents

1 Introduction

Wireless communication provides many benefits over using wired, lower infrastructure costs and the ability to easily move wireless transmitters being two reasons. However this comes at a tradeoff of speed and reliability.

This project is to create a program to simulate sending data through nodes that are grouped together into clusters. The clusters will be arranged linearly between the sender and the receiver as shown in Figure 1. The nodes in the clusters will be able to both send and receive data. The goal is to get all of the packets of data from the sender to the receiver by forwarding the packets along the clusters.

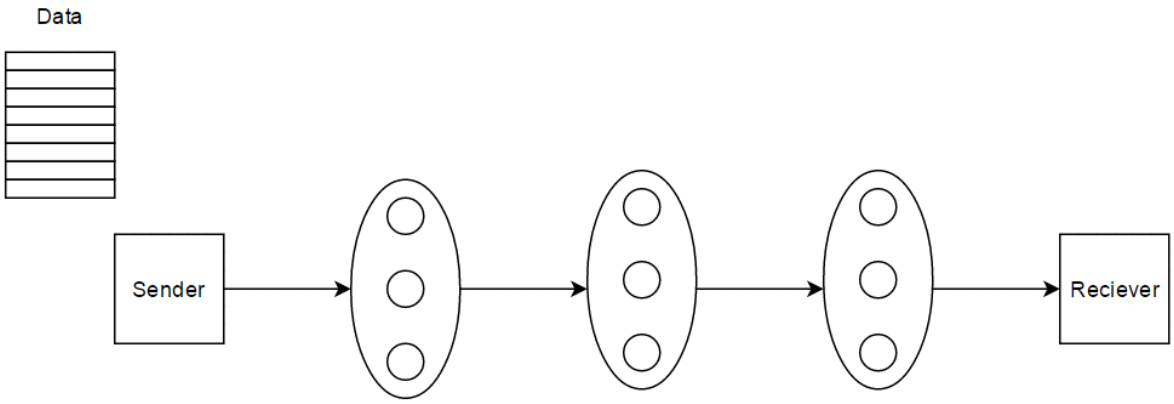


Figure 1. An example setup of three clusters each containing three nodes

The system is considered to be one-way, messages can only travel from the sender to the receiver. This removes the option of the receiver being able to send acknowledgements back to the sender to confirm that a packet has been successfully transmitted across the clusters. The purpose of having the nodes arranged in clusters is to increase the chance of a message being received. Only one node in a cluster will need to receive a packet for it to be able to continue along the clusters.

Noise on the wireless medium can cause bit errors in the reception of packets resulting in the packet being dropped. Forward error correction (FEC) can be used in wireless networks to protect against a small number of bit errors. The goal of the future research is to change the backoff distributions to try and reduce the number of packets lost to collisions. When a collision occurs, a large chunk of a transmission is lost. This is not something that FEC will be able to recover the data from. FEC will not be included in this project.

All of the wireless nodes in the system (including the sender and receiver) will use the CSMA protocol. This protocol uses back off timers when it detects the medium is busy and it needs to wait. The time it will wait is random and drawn from a probability distribution. By altering the probability distribution that this wait time is drawn from it may be possible for the performance of the network to increase. The distributions to investigate and the metrics that will be used to judge the performance are out of the scope of this project. This project is focused on creating a simulation tool that can be used to carry out this research.

There are two values that we are mostly interested in. The average time it takes for the receiver to get all of the packets and the average success rate. The contents of the packet does not matter, the size will be the only factor we take into account. It is assumed that the contents of the packet will contain some form of order number so we do not need to be concerned about ensuring the packets arrive at the receiver in order.

2 Background

2.1 Wireless Transmission

Wireless devices work by using an antenna to radiate electromagnetic energy into its surroundings. Antennas also convert this electromagnetic energy back to electrical energy with the same efficiency [2]. Not all antennas radiate the signal equally in all directions. Modeling directional antennas is out of the scope of this project, instead only omnidirectional antennas shall be considered. As the distance between the sender and the receiver grows, the strength of the received signal falls. This is known as the path loss. In a real world scenario wireless signals can reflect off of buildings and other surfaces, creating several paths. This leads to constructive and destructive interference based on the phase shift of the reflected signals. Modeling this is also out of scope of this project, instead the system is considered to have no environment around it. The free space path loss model is very simple and is all that will be required to model the path loss [3].

There are many different ways that the data of a packet can be modulated onto an electromagnetic signal. The different modulation techniques have differences in the bit error rate (BER) as shown in Figure 2. These differences will allow some modulation techniques to have a higher chance of still decoding the packet when handling collisions or path loss.

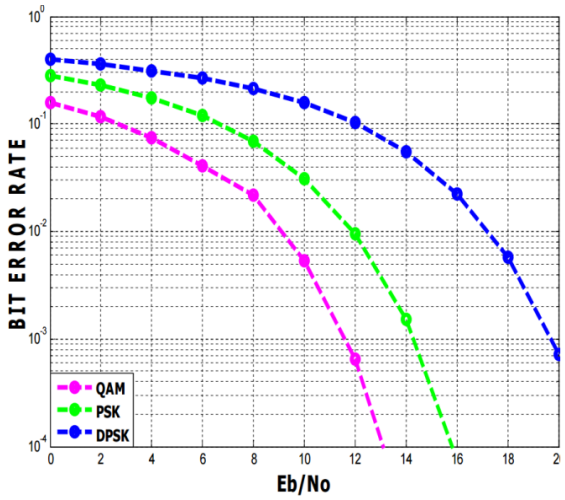


Figure 2. Comparing BER for SNR values of different modulation techniques [1]

Thermal noise occurs in the wireless transceivers due to the electrons moving within the condiuctive material of the circuitry. The electrons are moving due to the temperature of their environment [2]. As the temperature rises the thermal noise increases. This noise cannot be eliminated from the system and will always be at a constant level if the temperature does not change. This thermal noise provides an upper

barrier for the range at which two wireless devices can communicate. As they get further away the signal will get weaker. Eventually the signal will be so weak that the thermal noise will distort it to be unrecognisable.

A wireless transceiver is half-duplex, this means that it cannot send and receive at the same time. When a packet needs to be sent, the transceiver has to switch from receive mode to send mode [4]. This process is not instantaneous and takes a small amount to time. This is known as the turnaround time.

2.2 CSMA

The CSMA protocol is commonly used in wireless local area networks. It is a distributed protocol and does not require any set up or communication between the devices. It operates by listening to the medium first before sending a packet. If the medium is busy then it draws a back off time from a back off distribution. If it fails the carrier sense too many times then the packet at the front of the queue is dropped, and the process starts again. Figure 3 shows this protocol as a state diagram.

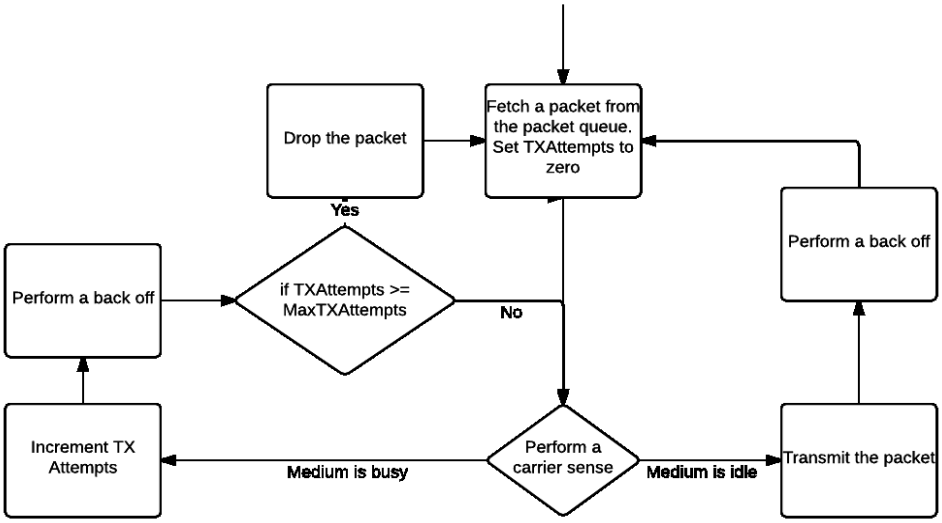


Figure 3. CSMA state diagram

In this project the CSMA protocol used does not use acknowledgement messages to very that the packet was successfully received. When the transceiver is in transmit state it can no longer listen to the medium. If a collision occurs, it cannot listen to the medium to detect it. Once the packet has been sent it can only assume that the transmission was successful.

2.3 Collisions

Even though the CSMA protocol checks if another transmission is in progress it is still possible for two transmissions to happen at the same time. As stated in section 2.1 it takes some time for the transceiver to switch to transmit mode. If during this time another sender performs a carrier sense it will also detect the medium as being idle, so it will transmit its own packet resulting in a collision.

Another situation where collisions can occur is the hidden terminal problem. This can have serious effects of the performance of a CSMA based wireless network [5]. This occurs when two nodes, which are out of range of each other, want to send a message to a receiver that is located in between them (shown in Figure 4). Sender A cannot hear sender B's transmission and so it sends its own packet. However by doing this it will cause a collision to happen at the receiver. To get around this issue a sender would need to perform a carrier sense at the location of the intended receiver. However this is not possible.

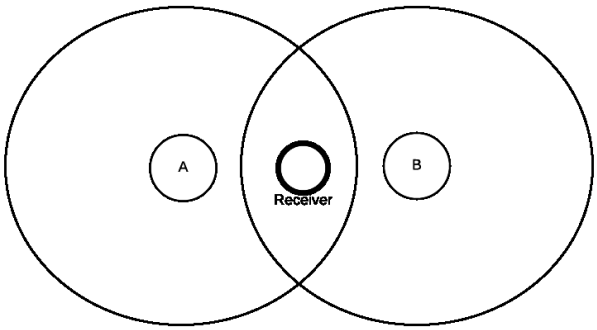


Figure 4. Diagram showing the hidden terminal problem

When a collision occurs the receiver sees the new signal as a large increase in the background noise [6]. If this increase is large enough then the receiver can choose to abandon the packet it was trying to receive, and switch to receiving the new packet that has significantly higher signal strength.

3 Solution

3.1 Frameworks

3.1.1 OMNeT++

OMNeT++ is a discrete time event based modeling framework. It was designed to be as generic as possible to allow for it to be used to model many different systems. Since its release in 1997 this idea has proven itself as it has now been used to build and simulate queuing systems to ad-hoc networks to business processes. One of the core principals of OMNeT++ is that the objects in the system should be designed to be modular allowing for as much reusability of components as possible [7]. To help achieve this all of the data that passes between modules must be wrapped in a message and go through defined gates. From a modules perspective where the output gates lead to is none of its concern nor is where the input gates come from. A module only needs to react accordingly to what messages it receives on its input gates. This allows for a user to be able to create their system by simply connecting the gates of their modules together.

A simulation in OMNeT++ is split into two different languages, C++ and the ned language. The ned language was created for OMNeT++ and is used to describe the topology of the simulation. The connections of a module, what parameters it takes, and the sub modules it is made up of. Upon launching the simulation OMNeT++ searches for a C++ class with the same name as the ned module. Having this decoupling between the topology and the implementation will prove useful as it allows for the topology to change between simulation runs without the need to compile code. One of the goals for the project is for the user to be able to change how a node generates the backoff time. This could be encapsulated into a module and used as a sub module to a node. This would allow for different back off algorithms to be used across the nodes without the need to compile their code in between configurations.

OMNeT++ was designed to be used by as broad of an audience as possible and does not provide any specialised classes or libraries. The OMNeT++ community has developed libraries that provide specialised functionality. For this project I will be using the MiXiM library.

3.1.2 MiXiM

MiXiM is a framework that focuses on simulating wireless networks. It provides modules for calculating the path loss, calculating bit errors, and handles sending the wireless signals [8]. It also provides modules that implement the CSMA protocol. For this project this framework should provide a good starting ground to build upon.

3.2 Design

3.2.1 OSI 7 Layer Model

The Open Systems Interconnection (OSI) model is a conceptual model designed to split the networking stack into layers to isolate it from layers that are not immediately above or below [9]. This abstraction allows for each layer to only need to be concerned with performing their task. The Application, Network, MAC (a sub layer of the Data Link layer), and Physical layer will be used to model the wireless nodes. Each of these layers are explained in more detail in the following sections.

The Presentation, Session, Transport, and Logical Link (a sub layer of the Data Link layer) layers are responsible for formatting data in the packet, managing sessions between users, creating and managing connections, and providing flow control and acknowledgements. These features are not needed for this project and have been excluded from the design of a node.

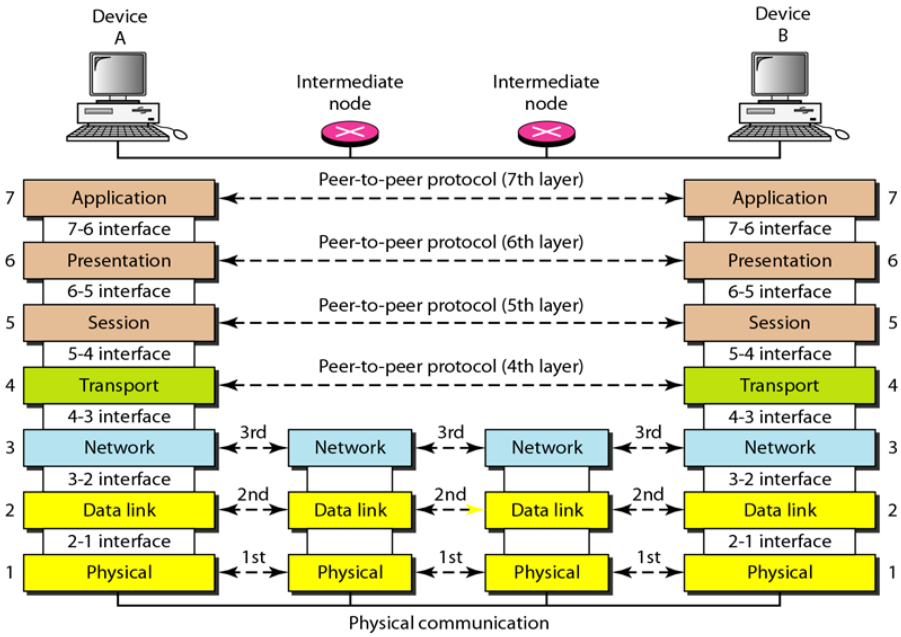


Figure 5. OSI seven layer model [10]

3.2.2 Module Design

3.2.2.1 Wireless Node

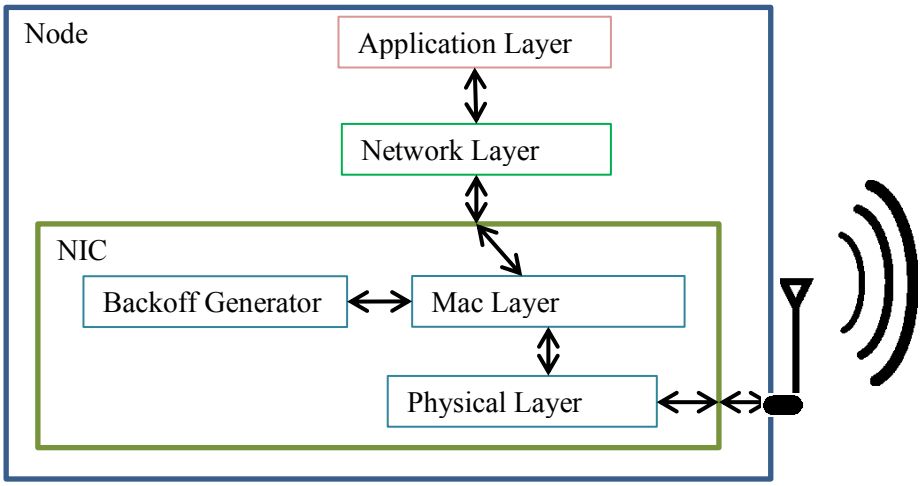


Figure 6. Wireless device design

The wireless sender, receiver and nodes all have the same module design as shown in Figure 6. Wireless device design above. For the four OSI layers that are used in this model, MiXiM provides a base module that can be extended. It also provides interfaces in the ned language for each layer specifying the gates. These interfaces allow for the implementation of a layer to be switched between nodes without needing to change how the gates are linked together. Having a common design that is used for all senders, receivers, and nodes allows for all of the gates to be defined and connected in a base node compound module. This compound module is extended for the sender, receiver, and node, which then defines what implementation of the application and network layer it wants to use.

The issue with using the module interfaces provided by MiXiM is that the network layer defines gates that are to be connected a higher layer. In the case of a node the network layer is conceptually the highest layer. OMNeT++ will not run a simulation that has unconnected gates. This requires there to be an application layer for the node even though according to the OSI model there should not be.

3.2.2.2 Application Layer

The Application layer on the sender will be responsible for creating the packets to send through the network. The delay between the creation time of the packets is passed as a parameter to the module. This allows for this time to be specified in the configuration file. This time could be drawn random number distributions to be a constant value. The receiver application layer will consume these packets and collect statistics. The nodes have been forced into having an application layer when it should not. If the node application layer receives a packet then an error message is printed and the simulation is terminated.

3.2.2.3 Network Layer

The Network layer is responsible for routing and forwarding packets. The network layer for the sender is very simple, it acts as a pass through between the application layer and the MAC layer.

For the node there is no need for any routing protocols due to the nodes not having a concept of the direction a packet needs to go in, they will just retransmit a packet that they receive. However it is not as simple as just sending back down any packet that arrives at the network layer. If we retransmit every packet that we receive, then two nodes will repeatedly broadcast the same packet to each other. A node should only retransmit a packet once. To do this it needs to know what nodes it has seen before. Each packet has the ID of the sender and the sequence number. A node needs to remember what packets it has seen from each sender. The most obvious solution is to store a list of all packet ID's seen. However while this solution would work well when there are very few senders, nodes and packets, it would fail for any large scale simulation. A simulation where there are three senders, each sending 1,000,000 packets across 50 nodes, would require $3 * 1000000 * 50 * 8 \approx 1GB$ of memory to store only the seen packet ID's. As the size of the simulation scales up, so does the memory required.

The packets are not guaranteed to arrive in order. By appending their ID's into a list will require that the list be searched each time a new packet arrives. This will cause a large reduction in performance as the list grows.

The sliding window protocol operates by having a lower and upper boundary on the sequence number of packets that it will accept [11]. As the window fills up by receiving packets, the window closes by moving the lower boundary up, and the opened again by increasing the upper boundary. This protocol will detect missing packets and will not move on until they have been received. The sliding window protocol has a low memory cost as it only needs to remember the sequence number of the boundaries, and what packets within that boundary have been received.

My solution for this takes inspiration from this protocol. A node has a list of size N for each sender. When a packet of ID P from that sender is received the index for that ID is found by calculating $P \ mod \ N$. The value at this index in the list is then retrieved. If it is less than P then the packet is new and the value is replaced by P. If it is equal or greater than P then the packet has been seen before and is dropped. This solution will work in the event of missing packets and has a $O(1)$ complexity for accessing and updating the seen list.

There is one case in which this method will fail. If, for instance, the list size is 10, then if a node receives packet 11 before it receives packet 1 then it will consider packet 1 to be a duplicate and drop it, even

though it has not seen it before. This could happen if a node repeatedly fails to get control of the physical medium. However the chance of this happening is very low, and can be further decreased by increasing the list size at the cost of using more memory.

For the receiver the network layer acts as a pass through like the sender network layer. To help collect statistics the receiver layer records how many times it has seen each packet from each sender. This allows for the 'reliability' of the network to be calculated. When a packet is sent this metric is the average number of times it is expected to arrive at the receiver.

3.2.2.4 MAC Layer

MiXiM provides a MAC layer that implements a CSMA protocol. This layer was extended and the back off function overwritten to allow for custom back off times to be used. When a back off time needs to be generated the MAC layer sends a message to the back off generator which then replies with the back off time.

Acknowledgement messages are commonly used with a wireless CSMA protocol as it allows for a sender to verify that the packet has been successfully received. When a carrier sense is performed that returns the medium as being idle, usually the MAC layer should wait for a small amount of time before sensing again then transmitting. This is done to stop a node causing collisions with the acknowledgement packet if it performs carrier sensing just after a transmission finished. The MAC layer provided by MiXiM implements this feature. This minimum wait time has been set to zero due to acknowledgements not being used in this project. While having the zero time wait does not change the results of the simulation, the overhead of having the send a message that performs no purpose will slow the runtime and should, in the future, be removed.

3.2.2.5 Physical Layer

The Physical layer is used to control how the data is transmitted over the medium. This layer focuses on how each packet is transmitted and received. MiXiM provides a physical layer that performs all of the tasks needed for this project. It keeps track of what signals are being sent and models collisions and bit errors. There are two helper classes that can be configured, the decider, and the path loss model.

The decider is responsible for generating the BER and deciding if the packet was received correctly. For this task MiXiM's 80211Decider was chosen, as it appeared to provide exactly what was needed. This decider had several different BER formulas based on the modulation used. In this project it is not critical what modulation formula is being used, as shown in Figure 2 the different techniques have the same curve

that is just shifted. This shifting could also be achieved by changing the distance between the nodes, or changing the transmit power.

3.2.2.6 Back off Generator

The back off generator module is very simple. It has only one input gate and one output gate which are connected to the MAC layer. The MAC layer sends a message to the back off generator when it wants a back off time. The back off generator responds with a message that contains this time. Having the back off generator as a separate module allows for the back off modules to be changed between simulation runs, by changing the configuration file, without needing to recompile the code.

3.3 Logging

Logging is a programming practice that has been widely used in commercial software development. It is a useful practice as it allows for the program to be able to communicate its state. This is especially useful when determining the location of defects or failures in the software. Log messages can also be used to gain additional statistics that are not output from the program. There is no rigorous specification on how logging should be performed. It is mostly up to the developer's discretion what they consider important enough to warrant a log message. This can be hard as logging a message comes with some performance overhead, so the frequency of that message should be considered.

Logging frameworks introduce the concept of assigning log messages with a verbosity level. When run the logging verbosity of the program is set, log message much be equal or above that level in order to be printed. This allows for the overhead of the logging to be removed when it is not needed. If the verbosity level is also printed it also allows for more important messages, such as errors, to be able to be searched for.

OMNeT++ and MiXiM currently log messages to the console, however they do not consistently follow a log message format. They also do not provide any logging classes, the messages are passed to an environment object that will accept strings and are printed to a console and a file. To aid in my design of my own logging class I looked into two object-oriented logging frameworks.

3.3.1.1 Log4J

Log4J [12] is a framework written in Java that I have experience with. For each class that logging is needed, a logger is instantiated with a name. This name is printed on the line with each message from that class, allowing for the origin of the message to be traced. Log4J has six verbosity levels, fatal, error, warning, information, debug, and verbose. Each verbosity level has its own function, rather than having one log function that takes a verbosity level as an argument.

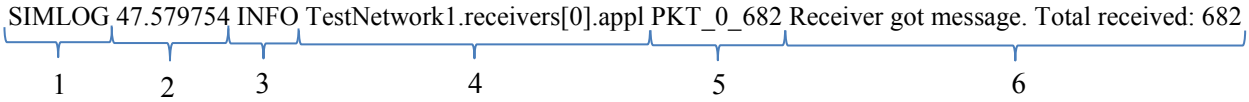
3.3.1.2 G2log

G2Log [13] is a very simple library written in C++11. Unlike log4J it does not use logging objects, it instead just uses a global logging function. The function takes the verbosity level as an enumerated value and the message to be logged. It does not provide a class name for the location of log message, instead it will have to be manually written into the log message. G2log uses four verbosity levels, fatal, warning, information, and debug.

3.3.1.3 Design

I chose to create a logger object similar to that used in the log4j framework as it makes it allows for the name of the owner to be saved and printed with each log message. Instead of passing in a name I chose to pass in the module. This causes the code to get the module name to only be in one place, rather than having to do in every module that uses a logger. I also chose to have the verbosity level as a parameter instead of having a function for each. This was to keep the code smaller, as I may need to make changes to it later.

When choosing a format for the log messages it was important to consider the ability for scripts to be able to separate out the parts of the message. Scripts usually use space characters to split parts of a line of text. There are six parts to a log message, each of which needs to be separated by a space and cannot contain a space (except for the last). The six fields are as follows:



1. The string “SIMLOG”. When the simulation runs OMNeT++ and MiXiM print their own logs to the console. This was added to allow for easy filtering to our own logs.
2. Simulation timestamp in seconds.
3. Verbosity level of the message.
4. Module that is logging the message.
5. The string “PKT_<SenderID>_<MessageNumber>”. This field is not always present. By adding a packet that was created by one of the senders as a parameter to the logger function it will add the sender id and packet number to the message. This can be used to filter on a particular packet so it can be traced.
6. The log message. This must go last as it will contain space characters.

4 Testing

There are many different types of automated testing that can be performed on software. Unit tests are very fine grained and aim at testing individual methods of a class. They test the logic contained within a class. In this project, most of the complicated logic and calculations are performed by classes in MiXiM. It is expected that third party libraries used in this project are tested by their developers. The code written for this project is not very logically complex. It is mostly connecting the MiXiM modules together while changing some behaviors. It was decided that unit tests would not be used as they would not provide enough benefit for the amount of effort required to create them.

Integration testing is the practice of combining modules together to see how they behave together. Integration testing suits this project very well. It is unfeasible to calculate a model results for a given set of parameters. Instead these integration tests will be used to test the behavior of the system as a whole. By changing an input parameter and inspecting the change in the results, behaviors of the simulation can be verified.

While the results of the test have to be manually inspected and interpreted, the running and collection of the results should be automated. After the simulation has been compiled, OMNeT++ produces an executable that can be started from the command line. Python scripts were used to start this executable and collect the relevant results from the output. Once the simulation runs have been completed the results are written into CSV files.

One feature of OMNeT++ is the way it generates its random numbers. When the simulation is launched the random number generator is always loaded with the same seed, resulting in the same 'random' numbers every time. While this feature is useful when debugging, the seed needs to be changed between test runs. This seed can set in the configuration file or passed via the command line argument. Python's own random number generator is used to create random number within the range of 0 to 2^{31} which is then used as the seed for the simulation run. Due to the stochastic nature of the simulation, when performing the tests, one simulation run with each set of parameters will not be enough to give a fair representation [14]. To give a fair representation of the behavior, each test is run 30 times with each set of parameters, and the results are averaged. A partial list of parameters that were used across all tests are given in Table 1. The packet generation time and back off distribution changes between each test. The packet length given only includes the size of the application layer packet. As the packet was send down the layer it was encapsulated and headers were added, increasing the final size.

Table 1. Simulation parameters

Parameter	Value
Radio turnaround time	0.25ms
Packet length	64 bytes
Thermal noise	-101dBm
Transmit power	50mW
Bit rate	250 kbps
Carrier frequency	2.4GHz
Max transmission attempts	5
MAC buffer size	5

4.1 Path Loss Test

Path loss is a very important part of this simulation. If it were not implemented correctly then either all nodes will not be able to receiver any message, or all nodes will be able to receive every message. This test only uses the sender and the receiver. The receiver does not send messages, leaving the sender to be the only node to be transmitting. This ensures that any packets that are not received correctly are due to a path loss and thermal noise, and not due to collisions.

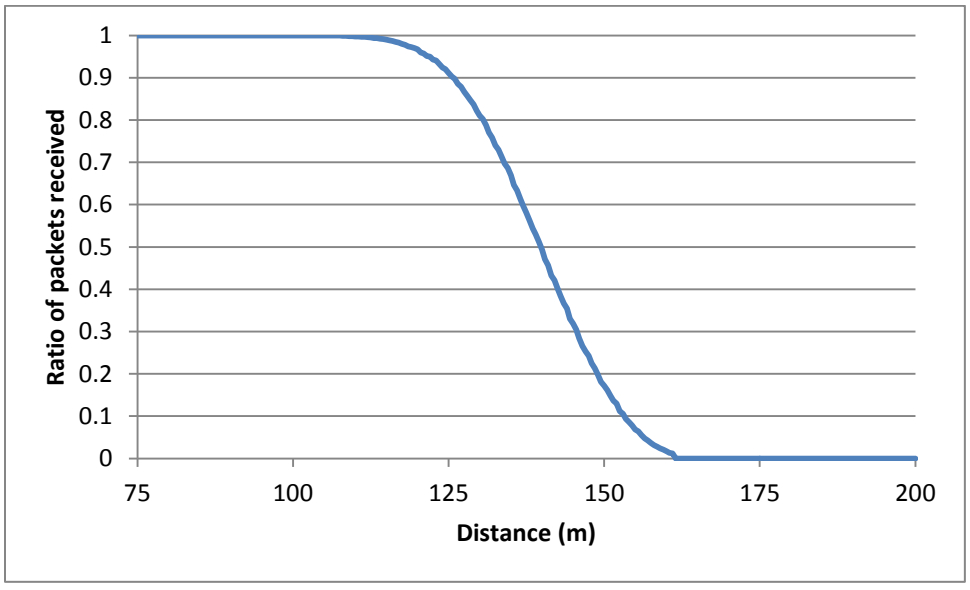


Figure 7. Path loss test results (average of 30 runs)

The results shown in Figure 7 are close to what is expected. The only issue is that when the packets receive rate is close to zero, instead of having a smooth curve, it jumps down to zero. When receiving a

packet MiXiM will first calculate the signal to noise ratio. The SNR is then compared to a threshold, if it is below then the packet is dropped without performing any bit error calculations. This threshold is set to 0.1, a ratio so low that there should be no chance that a packet can be received. However the drop off in the results would above indicate that this threshold is being reached. These results show that path loss is being calculated, thermal noise is also being calculated, but there is an issue with the bit error rate calculations.

4.2 Collision Test

During any simulation run it is expected that there will be collisions. To determine if collisions are resulting in packet loss this test was created. This test has one receiver and several senders surrounding the receiver. The senders are arranged in a circle with a radius of one metre. They are placed close to ensure that no packets will be lost due to path loss. In this scenario collisions will occur if two senders perform a carrier sense within the time it takes for the radio to turnaround. As more senders are added, the likelihood of this happening increases. As we increase the number of senders, the ratio packets received to packets should decrease.

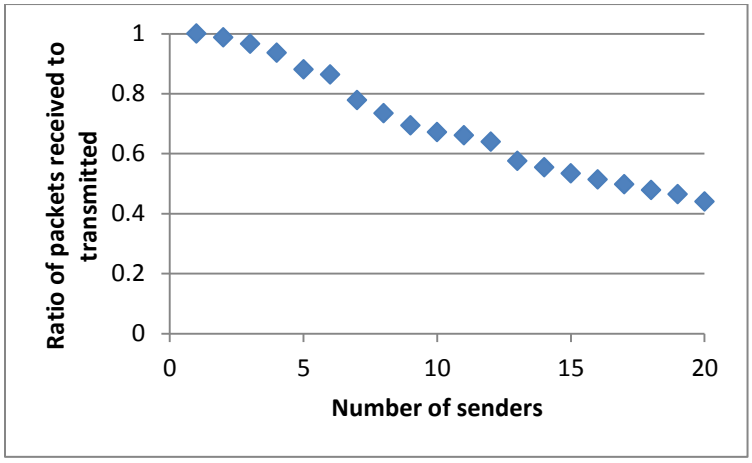


Figure 8. Results of collision test showing a decrease in successful transmissions

As shown in Figure 8 above, as the number of senders is increased, the chance of a sender transmitting a packet that is successfully received decreases. These results do not completely align with what is expected. The expected trend would be a curve that asymptotically approaches zero. This curve can be seen however it is quite linear. Let p be the probability of a sender having its back off timer expire during the turnaround time for another sender, which will result in a collision. When there are multiple senders in the system, it only takes one of them to have their back off timer expire during this turnaround time before the packets will be lost. The probability of a packet successfully being transmitted is given by $(1-P)^n$ where n is the number of other senders in the system. This should result in the ratio of packets

received to transmitted decrease rapidly at the start, then asymptotically approach zero as the number of senders approaches infinity.

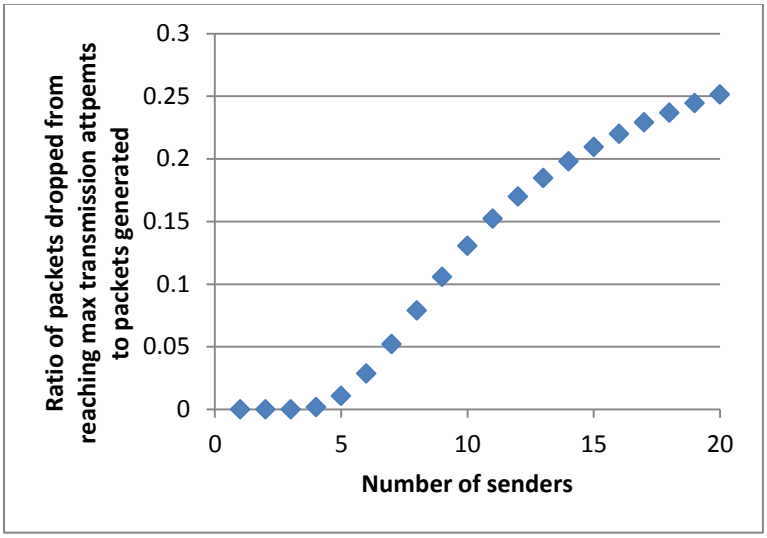


Figure 9. Results of collision test showing the packets dropped from reaching the maximum number of transmission attempts

As the number of senders increases so does the chance that the medium will be busy when a sender wants to send a packet. After a carrier sense has determined that the medium was busy five times then the packet will be dropped. Figure 9 shows that at a low number of senders there is very little chance that the carrier sense will fail five times resulting in no packets being dropped from reaching the maximum transmission attempts. After five senders have been added the medium is now busy enough that there is a significant chance that the carrier sense will fail five times. The graph appears to be approaching an asymptote at around 0.3. Even if there are very many senders there will always be a period of time that the medium is idle. After a transmission ends even if another sender performs a carrier sense immediately, the medium will remain idle until that sender's radio has switched to transmit mode.

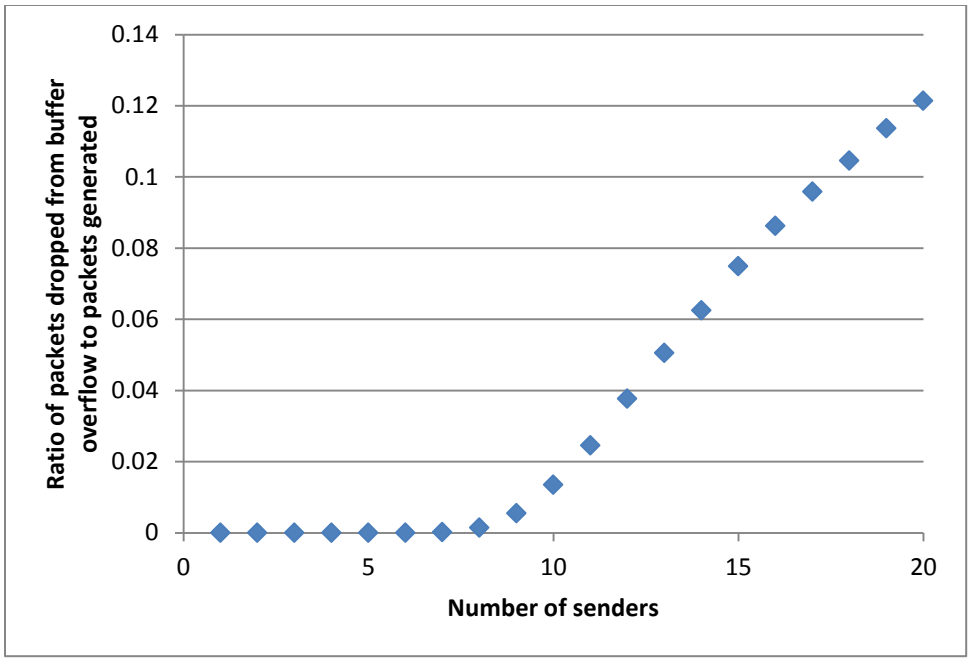


Figure 10. Results from collision test showing the packets dropped due to a buffer overflow

In the MAC layer the buffer is set to hold at most five packets. This is intentionally set to be small so it is more likely that it will overflow. When performing the collision test it can be seen in the graph above that there are a negligible number of packets dropped until eight senders are used. As the number of sender increases it the average number of back offs performed to send a packet increases. This results in packets sitting in the queue for longer, making it more likely that the queue will be full when a new packet arrives. The results shown in Figure 10 match this theory.

4.3 Near Far Test

Not all collisions will result in both packets being lost. If one of the signals is many times as loud as the other, then the stronger signal should drown out the weaker and still be able to be received. This test is to verify that other signals are being included in the noise calculations.

To have collisions we need to have two senders. One will be placed close to the receiver, and the other will initially be placed close, but will be slowly moved away. The packet generation time and the back off time are set to fixed values. This will cause the senders to stay synchronized with each other when they perform a carrier sense and transmit packets. Every packet will collide with the other when they arrive at the receiver. Initially while both nodes are close it is expected that both packets should be dropped as the noise will be too high. As the moveable sender is moved farther away, the noise when receiving the close sender packets will decrease, resulting in an increase in chance for the packet to be successfully received.

When the simulation was run, it did not return the results that were expected. At all distances for the moving sender no packets were received. However the close sender had all of its packets received, even when the moving sender was very close. When running the simulation in a debugger, it was seen that the SNR was close to 1 when the moving sender is close. This shows that the second transmission is being calculated in the noise values. As shown in the path loss test there is an issue with the BER calculation. Even when the SNR was close to 0.1 packets were still being received. This would explain why packets were still being received even though there was a collision. This could also be the cause of the unexpected results in from the collision test shown in Figure 8.

MiXiM's physical layer will only allow the receiver to attempt to receive one message at a time. The near sender was specified in the configuration as having index zero in the global list of senders. Both senders are synchronised perfectly, making the near senders events get processed first in the event queue. This would explain why the far sender would not have its packet received even though, with the current error in the BER calculation, it could be received.

4.4 Comparing Back off Distribution Test

The goal of the future research for this project is to investigate the usage of different backoff distributions for each node to determine if performance improvements can be gained. This test is designed to take two different distributions to see if there is an observable difference in the performance. The topology of this experiment is shown in Figure 1. The node groups are placed 75 metres apart. The nodes within a group are spaced 10 metres apart.

The first distribution model is very basic. It has all nodes draw its back off times from the same exponential distribution. The second distribution model still uses this exponential distribution but the time that is returned is then scaled based on the group that the node is a part of. For the nodes in the first group the back off time is unchanged, for the second group the time is reduced by a quarter, and for the third group the time is reduced by half. This will give the nodes in later groups a higher change of winning the contention of the medium.

Three metrics are collected and compared between the two different distribution schemes. The sender is configured to send 10,000 packets. The metrics collected are, the number of packets received at the receiver, the time it takes for the system to transport 10,000 packets from the sender to the receiver, and the average time it takes for one packet to arrive at the receiver from when it was created at the sender.

Table 2. Results comparing two distribution models (average of 30 runs)

	Basic Model	Scaled Model
Packets Generated	10000	10000
Packets Received	7162	7231
Total Runtime	200 seconds	200 seconds
Average Single Packet Time	53ms	46ms

As can be seen in Table 2 above, the scaled model had slightly more packets received, as well as having each packet takes less time to travel from the sender to the receiver. Some of this time decrease would be due to the lower average times for the back off on the second and third node group. However this lower time would only account for 4ms (a quarter reduction on the second group, and a half reduction on the third, with the average back off time being 5ms), leaving there to be a slight improvement.

To switch between using the two back off distributions is very easy. It only requires one line changed in the configuration file as shown in Figure 11.

```
162 **.nic.backoffType = "BasicBackoff"
163 **.nic.backoffType = "TaperedBackoff"
```

Figure 11. The configuration lines to switch between the back off models

5 Known Issues

5.1 MAC Reaching a Bad State

There is an issue that occurs when the back off time is smaller than the turnaround time of the transceiver. When a node has finished sending a packet the physical layer starts to switch back to receive mode and sends a message to the MAC layer to notify it that the transmission is finished. The CSMA mac layer provided by MiXiM schedules a back off before it tries to send the next packet. If this back off time is smaller than the turnaround time then the MAC layer will attempt to perform a carrier sense when the physical layer is still switching to receive mode. The MAC layer switches its state to carrier sense then checks if the physical layer is in receive mode before requesting if the medium is busy. There is no code to handle the case where the physical layer is not in receive state, leaving the method to exit without changing the state of the MAC. Next time the back off timer is triggered the MAC will still be in this carrier sense state instead of being in the receive state. In this case an error is printed to console and the simulation is terminated. To work around this issue whenever a back off time is generated it is checked to see if it is smaller than the turnaround time. If it is smaller, a message is printed and the back off time is changed to be slightly larger than the turnaround time. For future work it would be better to handle this case in the MAC layer code rather than relying on the back off time always being large enough.

5.2 BER Calculations

As was shown in 4.1 and 4.3 there is an issue with the BER calculations. The BER is being calculated far too low as resulted in packets being received that should not have been. It is clear that transmissions that have a SNR well below one are not being dropped. The formulas used by MiXiM will need to be checked to see if they are the cause of the issue.

6 Conclusion

The goal of this project was to create a software simulation of a wireless network. The wireless nodes should use a CSMA MAC protocol and should be able to have their back off distributions changed so the performance differences can be studied for future research. The OMNeT++ simulation framework was chosen and used to build the simulation in. The MiXiM framework was also used to provide much of the functionality that was needed to be able to build the simulation. However during implementation bugs in the MiXiM framework slowed down progress and has left the simulation with errors. The test cases highlight these flaws well. The path loss test and the near far test show that the bit error rate is being calculated to be too low, allowing packets to be received that should have been dropped. Before the simulation could be used for research purposes these issues need to be resolved. The design of the wireless devices allows for the back off distribution to be changed easily between simulation runs. Having it specified in the configuration file allows for the distributions to be assigned based on a nodes group number, or even assigned individually.

7 References

[1] s. kumar and S. Sharma, "Error Probability of Different Modulation Schemes for OFDM based WLAN standard IEEE 802.11a," *International Journal of Engineering,* vol. 4, pp. 262-267, 2010.

[2] W. Stallings, *Wireless communications and networks*. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[3] B. Sklar, "Rayleigh fading channels in mobile digital communication systems. I. Characterization," *IEEE Communications Magazine,* vol. 35, pp. 136-146, 1997.

[4] E. E. Johnson, M. Balakrishnan, and Z. Tang, "Impact of turnaround time on wireless MAC protocols," pp. 375-381 Vol.1.

[5] F. Liu, J. Lin, Z. Tao, T. Korakis, E. Erkip, and S. Panwar, "The Hidden Cost of Hidden Terminals," pp. 1-6.

[6] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein, "Overhaul of ieee 802.11 modeling and simulation in ns-2," pp. 159-168.

[7] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European simulation multiconference (ESM'2001)*, 2001, p. 65.

[8] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. K. Haneveld, T. E. V. Parker*, et al.*, "Simulating wireless and mobile networks in OMNeT++ the MiXiM vision," presented at the Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, Marseille, France, 2008.

[9] H. Zimmermann, "OSI reference model--The ISO model of architecture for open systems interconnection," *Communications, IEEE Transactions on,* vol. 28, pp. 425-432, 1980.

[10] E. E.-D. Ahmed, "The OSI Reference Model and Seven Layers of OSI," ed, 2012.

[11] D. Hercog, "Generalization of the basic sliding window protocol," *International Journal of Communication Systems,* vol. 18, pp. 57-75, 2005.

[12] *Apache Log4j 2*. Available: http://logging.apache.org/log4j/2.x/

[13] *g2log: An efficient asynchronous logger using C++11*. Available: http://www.codeproject.com/Articles/288827/g-log-An-efficient-asynchronous-logger-using-Cplus

[14] F. E. Ritter, M. J. Schoelles, K. S. Quigley, and L. C. Klein, "Determining the number of simulation runs: Treating simulations as theories by not sampling their behavior," in *Human-in-the-Loop Simulations*, ed: Springer, 2011, pp. 97-116.